UN MODELO DE CODIFICACION DINAMICA DEL METODO DE HUFFMAN, PARA LA COMPRESION DE DATOS EN LINEA

Autores: Lic. Luis Gabriel Rueda
Prog. Manuel Oscar Ortega
Ing. Raul Oscar Klenzi
Prog. Nelson Ruben Rodriguez

Docentes del Departamento Informática, Facultad de Ciencias Exactas Físicas y Naturales.
Domicilio:Ignacio de la Roza y Meglioli (5400) San Juan.
Teléfono: 064-231945; Fax: 064-234980
Mail: lrueda @iinfo.unsj.edu.ar

RESUMEN

Este trabajo consta de tres partes. La primera presenta un modelo teórico propuesto del método de codificación de Huffman Dinámico, en el que se enuncian dos teoremas funamentales. Luego un modelo de implementación del mismo, que muestra como puede llevarse a código de computadora el problema planteado en el modelo teórico, y por último las estructuras del algoritmo en Turbo Pascal y un ejemplo práctico de aplicación, que consiste en la codificación de una frase, paso a paso.

1.INTRODUCCION

Los métodos estáticos de codificación presentan una sencilla y rápida aplicación, debido a la simplicidad de sus algoritmos. El problema de estos es que previo a su aplicación, debemos tener las probabilidades de ocurrencia de cada uno de los símbolos. Pero en el caso de las comunicaciones, por ejemplo, donde no se cuenta con esta información, puede resolverse con el uso de los **métodos dinámicos**, en los cuales, voy leyendo, calculando las probabilidades de ocurrencia y codificando simultáneamente.

2.MODELO TEORICO

La condición necesaria para la codificación es la obtención de un código **instantáneo** o **prefijo**. Este puede implementarse fácilmente mediante una estructura de árbol de características especiales.

Definición 1: Sea $F = \{S_1, S_2, \ldots, S_q\}$, una fuente, se define como árbol de codificación en base-r A_r, al que cumple con las siguientes propiedades:

1) A_r es r-ario extendido (cada nodo tiene 0 o r hijos).

2) Cada nodo interno es la suma de los pesos de sus hijos correspondientes.

3) Cada hoja (nodo externo) se corresponde con uno y solo un símbolo y su peso respectivo.

Definición 2: Sea $F = \{S_1, S_2, \ldots, S_q\}$ una fuente de símbolos, y sea $P = \{P_1, P_2, \ldots, P_q\}$ las probabilidades de ocurrencia de cada símbolo, y $L = \{L_1, L_2, \ldots, L_q\}$ las longitudes de cada símbolo en un código, se define la longitud promedio L de un código como:

$$L = \sum_{i=1}^{q} P_i L_i$$

donde q es la cantidad de símbolos distintos de F.

En el transcurso de la codificación parece razonable el hecho de ir actualizando los pesos de cada uno de los nodos de A_r, pero más razonable sería que, además de ir actualizando los pesos, se fuera corrigiendo el árbol para que genere un código compacto, es decir de **longitud promedio mínima (L)**, y en consecuencia, mantener en forma permanente un A_r **Huffman**. Generar un nuevo árbol cada vez que un símbolo de la fuente arriba, lo convierte en un método totalmente ineficiente.
Ahora bien, veamos que condiciones debe cumplir A_r para que sea un A_r **Huffman**.

Teorema 1: (de la L mínima).
Sea A_r un árbol de codificación se dice que es un A_r **Huffman** de L **mínima**, sii:

$\forall$ nivel de A_r, $\forall$ n_i (nodo del nivel i), y $\forall$ n_j (nodo del nivel j), tales que $i < j$, se verifica:

$$Pn_i \geq Pn_j$$

donde: Pn_i es el peso de n_i ; y Pn_j es el peso de n_j

El Teorema anterior puede observarse en el desarrollo del metodo **estático**. Cada lista nueva de pesos que se genera, se reordena y se suman los últimos r pesos, es decir, los de menor valor, para obtener uno solo en la lista siguiente; lo que significa que estos últimos, en el proceso de asignación de códigos, suman 1 a su longitud, lo que implica que en el árbol deban estar en el mismo nivel, o más abajo, que el resto de los elementos que no se sumaron.

Definición 3: Sea $F = \{S_1, S_2, \ldots, S_q\}$ una fuente , con $P = \{P_1, P_2, \ldots, P_q\}$ las probabilidades de ocurrencia de cada símbolo, y $L = \{L_1, L_2, \ldots, L_q\}$ las longitudes de cada símbolo en un código, se define la varianza V de un código como:

$$V = \sum_{i=1}^{q} P_i (L_i - L)^2$$

Teorema 2: (de la V mínima).
Sea A_r un árbol de codificación se dice que es un A_r **Huffman** con V **mínima** sii, cumple con el **Teorema 1**, y ademas:

$\forall$ nivel de A_r, $\forall$ n_i (nodo del nivel i), y $\forall$ n_j (nodo del nivel j), tales que $i < j$, si sucede que:

$$Pn_i = Pn_j$$

se cumple: $$cn_i \leq cn_j$$

donde: cn_i es la cantidad de hijos de n_i ; y
cn_j es la cantidad de hijos de n_j

El teorema anterior puede verse, al igual que el **Teorema 1**, en el desarrollo del **estático**, si al formar una nueva lista, el elemento obtenido como suma de los r anteriores, se lo ubica arriba de los que tienen igual peso que este, tiene menos posibilidad de ser sumado en el paso siguiente, y en consecuencia, en el proceso de asignación de códigos, no sumara 1 a su longitud, y queda en el mismo nivel que los de su mismo peso.

La necesidad de obtener un árbol que cumpla con los **Teoremas 1** y **2**, hace que se realicen cambios entre distintos nodos, ya que luego de ingresar varios símbolos de la secuencia de entrada, este se desordena y se hace dificil para trabajarlo, por ej. para buscar un símbolo.

Definición 4: Sea $C = \{X_1, X_2, \ldots, X_q\}$ un **código** para una fuente F, se dice que es **ordenado**, sii verifica la siguiente propiedad:

- Sean S_i y S_j, dos símbolos de la fuente, con P_i y P_j sus correspondientes probabilades de ocurrencia, X_i y X_j las palabras código; para todo i,j, tal que $P_i > P_j$:

$$X_i < X_j$$

Esto es, si un símbolo tiene mayor probabilidad que otro, su palabra código es menor que la del segundo; por ejemplo, la palabra código en base **2**: ***10010*** es menor que ***100110***.

Luego diremos que A_r es un **Arbol Ordenado**, si su C correspondiente cumple con la definición anterior.

El método de codificación de **Huffman Estático**, generalmente produce un código desordenado, dependiendo de la asignación de los símbolos del alfabeto código, por lo tanto su árbol no siempre es ordenado. Existen otros metodos estáticos como el de **Shannon** o el de **Fano**, que en todos los casos generan un C **ordenado**.

Ahora bien, sabemos que A_r es ordenado si cumple con la **def.4**, lo que significa que las hojas (correspondientes a cada símbolo) estan ordenadas de izquierda a derecha según su peso. Si tomamos la base 2, cualquier cambio entre dos nodos del mismo padre no afecta para nada la longitud promedio (pues sigue cumpliendo con el **Teorema 1**).

Comencemos por la raíz, cambiando sus dos hijos, tenemos dos árboles distintos, luego por cada uno de ellos, tenemos dos más, es decir dos por dos (cuatro), y así sucesivamente, para cada nodo interno.

Definición 5: Sea A_r un árbol huffman, y sea n la cantidad de nodos internos de A_r , la cantidad de nodos de A_r se define como de 2^n.
Si tenemos una fuente con q símbolos, cuanto vale n es decir cuantos nodos internos tiene A_r?
Tengo $q-1$ nodos internos, es decir $n=q-1$, entonces:

Definición 6: Sea A_r un árbol huffman, y sea $F = \{S_1, S_2, \ldots, S_q\}$ una fuente, la cantidad de nodos de A_r se define como de 2^{q-1}.

3. MODELO DE IMPLEMENTACIÓN

3.1. Codificación

El método propuesto presenta una forma de codificación dinámica que cumple con el Teorema 1, es decir, con L mínima.
Como puede observarse, a la hora de implementar este método no es eficiente desde el punto de vista computacional, basta darse cuenta imaginándose de que cada vez que un nodo se actualiza debe ser comparado con todos los nodos de los niveles anteriores.
Este problema puede solucionarse si mantenemos, además del árbol de codificación, una lista lógica, que comienza desde el nodo de mas a la derecha del último nivel hacia la izquierda, subiendo de nivel hasta llegar a la raíz del árbol. Esto supone un enlace de los nodos del árbol en ese orden (ya sea a través de punteros, o un árbol implementado con arreglo ordenado por pesos).
Imaginemos la lista ordenada de izquierda a derecha, de menor a mayor por pesos.
Lógicamente, el codificador y el decodificador están de acuerdo en usar el mismo método.
Para la implementación del método, es necesario llevar un símbolo imaginario i, que representa el próximo símbolo desconocido de la fuente.
Si tomamos como referencia la base binaria (0 y 1), (puede extenderse a una base $r > 2$), para cada símbolo que arriba:

a) Lo busco en la lista, si no está, creo un nuevo subárbol, a partir del nodo del símbolo i con peso 1, y sus hijos son el nuevo elemento con peso 1 y el símbolo i con peso 0.

b) Comienzo a caminar por el árbol hacia arriba, enviando a la salida un 0 si el hijo es el izquierdo o un 1 si es el derecho.

c) Cada nodo n_i visitado, es intercambiado por el de elemento de más a la derecha en la lista que tiene peso menor que n_i.

d) Si el símbolo que arribó es nuevo, luego de enviar el código, envío éste a la salida, para su reconocimiento en la decodificación.

Para acelerar la busqueda del paso a), podemos usar una técnica de **hashing**, llevando una tabla adicional que contenga para cada símbolo la dirección en la lista enlazada (por ejemplo, si codificamos con el alfabeto **ASCII** como símbolos independientes, puede llevarse una tabla de 256 posiciones, cada una subindicada por el caracter correspondiente)

3.2. Decodificación

El metodo de decodificación se implementa de la misma manera que el de codificación, pero en este caso se comienza a caminar desde la raíz hacia las hojas del árbol, hacia la izquierda (0), o a la derecha (1), hasta encontrar un símbolo.

Para una fuente finita, el último símbolo nuevo que arribó, puede ser reemplazado por ***i***, lo que implica un ahorro en ***L***, pues disminuye en 1 la longitud de éste; además se produce una pequeña reducción en el espacio de búsqueda.

4. APLICACIÓN

```
type
  s   = string[15];
  s12 = string[12];
  FCIA = record
          caracter : char;
          fcia     : word;
        end;
type
{ T.D.A. ÁRBOL de Codificación }
  ÁRBOL  = ^NODO_ÁRBOL;
  NODO_ÁRBOL = record
                 elem     : FCIA;
                 izq,             { Puntero al Hijo Izquierdo  }
                 der,             { Puntero al Hijo Derecho  }
                 padre    : ÁRBOL; { Puntero al Padre  }
               end;
type
{ T.D.A. ARREGLO DE CARACTERES }
  NODO_ARREGLO = record
                   esta: boolean;
                   ptro: ÁRBOL;
                 end;
  ARREGLO_CAR = array[#0..#255] of nodo_arreglo;
type
{ T.D.A. LISTA (FORESTA) }
  LISTA = ^NODO_LISTA;
  NODO_LISTA = record
                 ptro_arb : ÁRBOL;
                 sig      : LISTA;
               end;
{ T.D.A. PILA (EN REEMPLAZO DE LA LISTA ANTERIOR) }
  PILA = record
           tope: word;
           datos: array[1..515] of ÁRBOL;
         end;
```

Figura 4.1 Estructuras del Algoritmo

4.1 Algoritmo

El algoritmo de codificación del método presentado en la sección 3.1, puede implementarse en un lenguaje de programación que soporte estructura de punteros, tales como Pascal, C. La razón de esto es la necesidad de llevar un árbol, y una lista enlazada de todos los nodos del mismo, en forma simultánea.
El algoritmo se ha implementado en Turbo Pascal 7.0. Las estructuras del mismo se muestran en la Figura 4.1. Puede observarse que cada nodo tiene un puntero a su hijo, y otro a su padre (para caminar hacia arriba).
Además, la estructura ARREGLO_CAR, que es la tabla de **hashing** para la búsqueda planteada en la sección 3.1

Se ha realizado una prueba del algoritmo para archivos en D.O.S., en los que se ha logrado mayor compresión que con el estático, pero, no del todo satisfactoria en cuanto a la velocidad de ejecución. Para ello se plantea la variante de usar arreglos en lugar de punteros, que inclusive, el problema sería mas claro en su representación.

4.2 Ejemplo

En esta sección se presenta un ejemplo práctico, que consiste en la codificación de la palabra ***abracadabra***, en base 2. Solamente se hará hasta el símbolo ***d***, porque en este punto se produce un cambio importante; el resto puede continuarse de manera análoga.
Cada nodo tiene dos casilleros, el de la izquierda contiene el símbolo (para las hojas) y el de la derecha representa el peso del mismo.

Nos introduzcamos en el ejemplo. A la salida se envía el símbolo ***a***, y simultáneamente se actualiza el árbol y la lista (Fig.4.2).

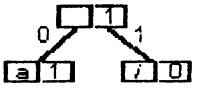

Figura 4.2

El próximo símbolo es ***b*** (Fig.4.3) como es nuevo se envía la codificación del símbolo ***i*** mas éste, o sea ***1b***, puede verse que no hay necesidad de hacer cambios.

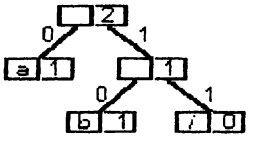

Figura 4.3

Cuando arriba el símbolo ***r*** (Fig.4.4), también es nuevo, a la salida se envía ***11r***, se intercambia el nodo *2* con a_1. Este cambio no es necerario para el ***Teorema 1***, pero obliga a ordenar la lista enlazada requerida en la implementación, que va desplazando los de mayor peso hacia la izquierda en el árbol, y en consecuencia están mas cerca del nivel superior

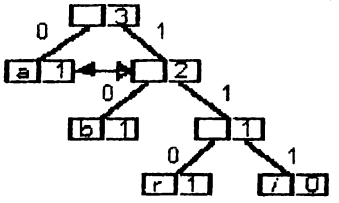

Figura 4.4

El símbolo ***a*** (Fig.4.5), no obliga a efectuar ningún cambio, la salida es ***1***.

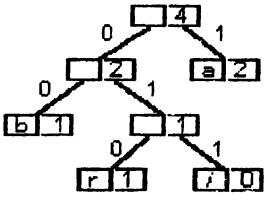

Figura 4.5

El símbolo ***c*** (Fig.4.6) es nuevo, su codificación es ***011c***, y se inserta a partir de i_0. Se debe intercambiar b_1 con ***2*** (esto no es necesario para el ***Teorema 1***).

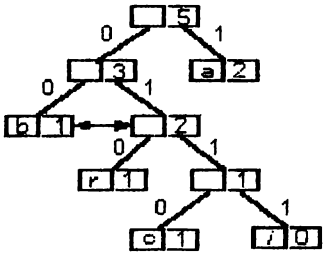

Figura 4.6

El símbolo ***a*** (Fig.4.7), no obliga a efectuar ningún cambio, la salida es ***1***.

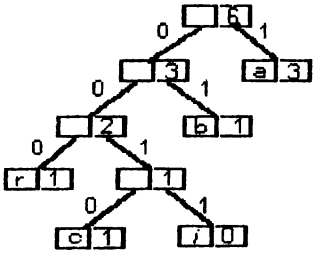

Figura 4.7

El ingreso de un nuevo símbolo ***d*** (Fig.4.8), obliga a realizar un cambio muy importante, (entre el nodo ***2*** y b_1), ya que es necesario para que A_r cumpla con el ***Teorema 1***. La salida será la de i_0 más el símbolo, esto es ***0011d***.

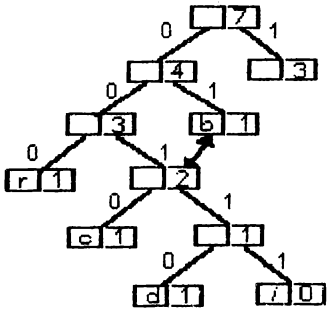

Figura 4.8

Si codificamos por Huffman estático, los símbolos que tenemos hasta este momento:

S_i	Cod.	Peso				
a	*1*	**3**	3	3	3	**4**
r	*000*	**1**	1	**2**	2	3
b	*001*	**1**	1	1	**2**	
c	*010*	**1**	1	1		
d	*0110*	**1**	**1**			
i	*0111*	**0**				

Observemos detenidamente el desarrollo del estático. Cada elemento de la columna de Pesos (resaltados), se corresponde con cada una de las hojas del árbol (Fig.4.9). Los elementos resaltados y subrayados, se corresponden con cada uno de los nodos internos del árbol, y justamente, estos se obtuvieron como la suma de los dos elementos de menor peso de la columna anterior.

Aquí se ve la esencia del ***Teorema 1***. De una columna a la siguiente debo elegir los dos elementos de menor peso, si así no lo hiciera, estaría aumentando en 1 la longitud de un símbolo de mayor peso, y disminuyendo en 1 la longitud de uno de menor peso. Obviamente, esto aumentaría L, y el código no sería **compacto**.

También puede observarse que, si tenemos en cuenta de que la rama izquierda simboliza un ***0*** y la derecha un ***1***, la codificación que se obtuvo en el estático es **exactamente** la misma que el árbol de la figura 4.9.

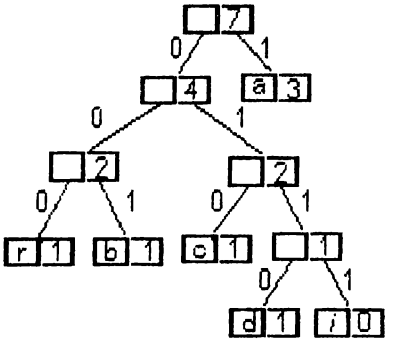

Figura 4.9

5. CONCLUSION

El algoritmo planteado, resuelve claramente un problema de comunicaciones, por ejemplo, solamente que se debería acelerar el procesamiento, para permitir codificar y decodificar en línea.

Otro de los aspectos importantes, es que ahora estamos en condiciones de intuir lo siguiente:
El metodo de Huffman dinámico, se resuelve a partir de la formación de cada uno de los estáticos en cada paso. Otros métodos estáticos, tales como Fano y Shannon,podrían ser implementados en forma dinámica?

La respuesta es sí, y bastará con mantener en línea la información necesaria acerca de los símbolos, digamos sus ocurrencias, y solamente codificar cada símbolo que arribe, e inmediatamente emitir su salida correspondiente.

6. BIBLIOGRAFIA

Abramson N., Teoría de la Información y Codificación, Addison Wesley.

Hamming Richard, Coding and Information Theory, Prentice Hall.

Lipschutz S., Estructura de Datos, Mc Graw Hill, 1988.

Storer J., Data Compression: Methods and Theory, Computer Science Press.

Van Lint, Introduction to Coding Theory, J. GMT 86.